Fast and Private Computation of Cardinality of Set Intersection and Union

Emiliano De Cristofaro[†], Paolo Gasti[‡], and Gene Tsudik[‡]

[†]PARC [‡]UC Irvine

Abstract. With massive amounts of electronic information stored, transferred, and shared every day, legitimate needs for sensitive information must be reconciled with natural privacy concerns. This motivates various cryptographic techniques for privacy-preserving information sharing, such as Private Set Intersection (PSI) and Private Set Union (PSU). Such techniques involve two parties – client and server – each with a private input set. At the end, client learns the intersection (or union) of the two respective sets, while server learns nothing. However, if desired functionality is private computation of *cardinality* rather than contents, of set intersection, PSI and PSU protocols are not appropriate, as they yield too much information. In this paper, we design an efficient cryptographic protocol for *Private Set Intersection Cardinality* (PSI-CA) that yields only the size of set intersection, with security in the presence of both semi-honest and malicious adversaries. To the best of our knowledge, it is the *first* protocol that achieves complexities *linear* in the size of input sets. We then show how the same protocol can be used to privately compute set union cardinality. We also design an extension that supports *authorization* of client input.

1 Introduction

Proliferation of, and growing reliance on, electronic information trigger the increase in the amount of sensitive data stored and processed in cyberspace. Consequently, there is a strong need for efficient cryptographic techniques that allow sharing information with privacy. Among these, Private Set Intersection (PSI) [11, 24, 14, 21, 15, 22, 9, 8, 18], and Private Set Union (PSU) [24, 15, 17, 12, 32] have attracted a lot of attention from the research community.

PSI allows one party (client) to compute the intersection of its set with that of another party (server), such that: (i) server learns nothing about client input, and (ii) client learns no information about server input, beyond the intersection. Efficient PSI protocols have been used as building blocks for many privacy-oriented applications, e.g., collaborative botnet detection [28], denial-of-service identification [1], on-line gaming [5], intelligence-community systems [19], social networks [25], and operations on human genomes [3].

Nonetheless, in certain information-sharing settings, PSI and PSU offer limited or no privacy to server. Consider the following scenario: after running PSI, the set intersection privately learned by client corresponds to entire server input – server privacy is non-existent, while client's is fully preserved. Similarly, after running PSU, set union privately obtained by client is such that its size is the

sum of the two respective set sizes: again, server looses all privacy since its entire input is disclosed to client. These examples illustrate the need for server to enforce a policy, based on the cardinality of set intersection/union, that governs whether it is willing to take part in PSI or PSU protocols.

1.1 Motivation

In general, the need for **Private Set Intersection Cardinality (PSI-CA)** and **Private Set Union Cardinality (PSU-CA)** protocols is motivated by a class of scenarios where client is *only allowed to learn the magnitude* – rather than the content – of set intersection/union. For instance, PSI-CA is useful in social networking, e.g., when two parties want to privately determine the number of common connections in order to decide whether or not to become friends. Also, as shown in [3], efficient PSI-CA protocols are instrumental to privacy-preserving genomic operations. Moreover, PSI-CA is useful to privately compare equal-size low-entropy vectors, e.g., to realize private computation of Hamming Distance between two strings. Indeed, two parties may use PSI-CA, by treating each bit, coupled with its position in a string, as a unique set element, such that client privately learns the number of elements (bits) in common, thereby computing the Hamming Distance. Other applications of PSI-CA include role-based association rule mining [23], location sharing [29], affiliation-hiding authentication [2], etc.

1.2 Contributions

We present efficient constructions for PSI-CA – a cryptographic primitive, involving server (on input of a private set $\mathcal{S}$) and client (on input of a private set $\mathcal{C}$), that results in client outputting $|\mathcal{S} \cap \mathcal{C}|$. Prior work has yielded some PSI-CA techniques (see Section 2), however, none is efficient enough to scale to large sets: current protocols involve a number of costly cryptographic operations (e.g., modular exponentiations) *quadratic* in the size of participants' sets, thus, their overhead is acceptable only for relatively small sets. Some constructs also incur quadratic *communication* overhead. Whereas, our PSI-CA technique is the first to offer **linear computation and communication** complexity, coupled with provable privacy guarantees. We provide two protocol "flavors": one secure in the semi-honest, and the other in the malicious, model. (The latter allows a semi-honest client to interact with a malicious server.)

We also present a protocol variant, called **Authorized PSI-CA (APSI-CA)**, where client input must be authorized (signed) by a mutually trusted authority (CA). Finally, we show how to combine PSI-CA with PSI such that server can decide whether to engage in PSI according to its policy based on the size of the intersection (privately obtained using PSI-CA).

Paper organization: Next section reviews related work. Section 3 presents our PSI-CA protocol, along with security proofs. Then, Section 4 constructs a protocol for APSI-CA, and Section 5 sketches a three-round policy-based PSI variant. The paper concludes with Section 6.

2 Related Work

Private Set Intersection and Union. [11] introduced the Private Set Intersection (PSI) problem and presented techniques based on Oblivious Polynomial Evaluations (OPE-s) and additively homomorphic encryption (e.g., Paillier [30]). The intuition is to represent a set as a polynomial and its elements – as the polynomial's roots. Client encrypts the coefficients, that are then evaluated homomorphically by server. As a result, client learns the intersection and nothing else. Assuming that server and client sets contain w and v items, respectively, client's computation complexity amounts to $O(w + v)$, and server's – $O(wv)$ exponentiations. [11] also proposes techniques to asymptotically reduce server workload to $O(w \log\log v)$ using Horner's rule and balanced bucket allocation. [15] obtained similar complexities while also offering PSU techniques. Whereas, [24] extended OPE-s to more than two players, all learning the intersection/union, with quadratic computational and linear communication complexities. Other PSI constructs, such as [14, 21], rely on Oblivious Pseudo-Random Functions (OPRF-s) and reduce computation overhead to a *linear* number of exponentiations. Recent results in the Random Oracle Model (ROM) led to very efficient PSI protocols, also with linear complexities while using much more efficient cryptographic tools. They replace OPRFs with unpredictable functions [22] and blind signatures [9], with security under *One-More-DH* and *One-More-RSA* assumptions [4], respectively. Finally, [8] achieved linear communication and computational complexities, using short exponents, with security in the malicious model, under the DDH assumption, while [18] shows a construction in the semi-honest model based on garbled circuits [35] which, leveraging so-called Oblivious Transfer Extension [20], scales relatively gracefully for very large security parameters.

Authorized Private Set Intersection. Authorization of client input was first investigated in [6] and [7], in both cases, with quadratic complexity. Authorized Private Set Intersection (APSI) was later formalized in [9] and [8] that constructed efficient techniques with linear complexity, secure under the RSA assumption. Recently, [33] proposed Policy-Enhanced PSI, allowing two parties to privately share information while enforcing complex policies. In this model, both parties' sets must be authorized, and both parties obtain the intersection.

Private Set Intersection Cardinality. Prior work yielded several PSI-CA protocols, although *none* with linear complexity:

- PSI protocol in [11] can be extended to PSI-CA with the same complexity, i.e., $O(w \log\log v)$ computation and $O(w + v)$ communication.
- [16] presented a PSI-CA protocol based on [11] with similar (sub-quadratic) complexities.
- [24] proposed a PSI-CA protocol for multiple ($n \geq 2$) parties, incurring $O(n^2 \cdot v)$ communication and $O(v^2)$ computational overhead.
- [34] constructed a multi-party PSI-CA protocol, based on commutative one-way hash functions [27] and Pohlig-Hellman encryption [31]. It incurs n rounds, and involves $O(n^2 \cdot v)$ communication and $O(vn)$ computational overhead.

- Finally, [6] presented an APSI variant (private intersection of certified sets) that computes the cardinality of (certified) set intersection and incurs quadratic communication and computation complexity.

3 New PSI-CA and PSU-CA

In this section, we define PSI-CA/PSU-CA functionalities, along with their privacy requirements, and present our constructs.

Definition 1 (Private Set Union Cardinality (PSU-CA)). *A protocol involving server, on input a set of w items $\mathcal{S} = \{s_1, \ldots, s_w\}$, and client, on input a set of v items $\mathcal{C} = \{c_1, \cdots, c_v\}$. It results in the latter outputting $|\boldsymbol{\mathcal{U}}|$, where: $\mathcal{U} = \mathcal{S} \cup \mathcal{C}$.*

Definition 2 (Private Set Intersection Cardinality (PSI-CA)). *A protocol involving server, on input a set of w items $\mathcal{S} = \{s_1, \ldots, s_w\}$, and client, on input a set of v items $\mathcal{C} = \{c_1, \cdots, c_v\}$. It results in the latter outputting $|\boldsymbol{\mathcal{I}}|$, where: $\mathcal{I} = \mathcal{S} \cap \mathcal{C}$.*

Informally, both PSI-CA and PSU-CA entail the following privacy requirements:

- *Server Privacy.* Client learns no information beyond: (1) cardinality of set intersection/union and (2) upper bound on the size of $\mathcal{S}$.
- *Client Privacy.* No information is leaked about client set $\mathcal{C}$, except an upper bound on its size.
- *Unlinkability.* Neither party can determine if any two instances of the protocol are related, i.e., executed on the same input by client or server, unless this can be inferred from the actual protocol output.

Remark: For any $\mathcal{C}$ and $\mathcal{S}$, the size of $\mathcal{C} \cup \mathcal{S}$ can be computed as $|\mathcal{C}| + |\mathcal{S}| - |\mathcal{C} \cap \mathcal{S}|$. Thus, privately computing cardinality of the *intersection* of $\mathcal{C}$ and $\mathcal{S}$ allows one to privately compute the cardinality of their *union* as well. Consequently, the rest of the paper only focuses on PSI-CA.

3.1 PSI-CA Secure in the Semi-Honest Model

The proposed PSI-CA protocol is shown in Figure 1. It is executed on common input of two primes p, q (where $q|p-1$), a generator g of a subgroup of size q, and two hash functions (modeled as random oracles), $H : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$ and $H' : \{0,1\}^* \rightarrow \{0,1\}^\kappa$, given the security parameter κ.

Intuition: First, client masks its set items (c_i-s) with a random exponent (R'_c) and sends resulting values (a_i-s) to server, which "blindly" exponentiates them with its own random value R'_s. *Server* shuffles the resulting values (a'_i-s) and sends them to client. Also, server receives, from client, $X = g^{Rc}$, a value resembling an ElGamal public key, and sends client the output of a one-way function, $H'(\cdot)$, computed over the product of X and exponentiations of server's items (s_j-s) to randomness R'_s. Finally, client receives server's "public-key like" value $Y = g^{R_s}$ and tries to match one-way function outputs received from server with one-way function outputs computed over the product of Y^{R_c} and the shuffled

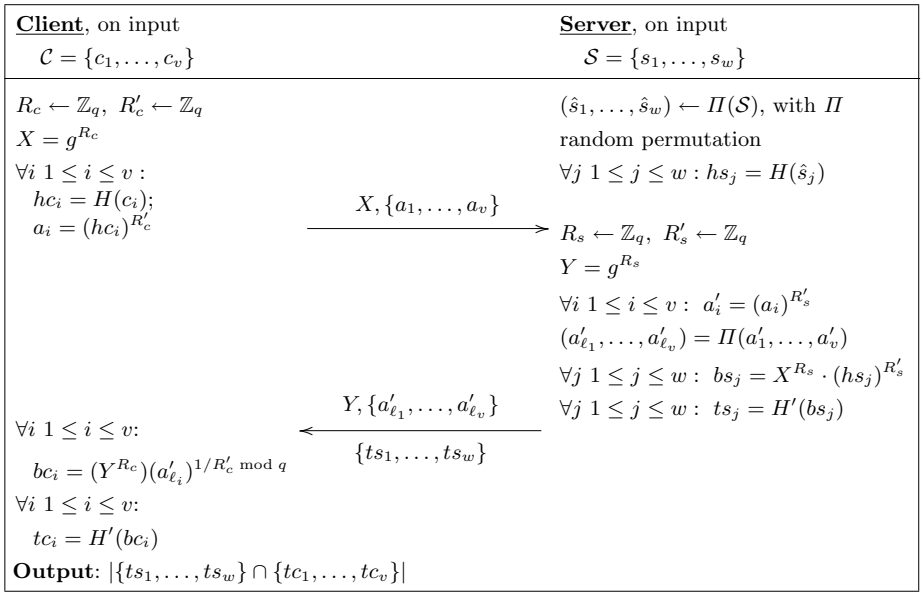

Figure 1: Proposed PSI-CA Protocol. All computation is mod p.

(a'_i-s) values, stripped of the initial randomness R'_c. *Client* learns the set intersection cardinality (and nothing else) by counting the number of such matches. As showed below, unless they corresponding to items in the intersection, one-way function outputs received from server cannot be used by client to learn related items in server's set (under the Gap-One-More-DH [22] assumption). Also, client does not learn *which* items are in the intersection as the matching occurs using *shuffled* a'_i values.

Complexity is *linear* in the sizes of the two sets. Let $|\mathcal{S}| = w$ and $|\mathcal{C}| = v$. Client performs $2(v+1)$ exponentiations with short, i.e., $|q|$-bit, exponents modulo $|p|$-bit and v modular multiplications. Server computes $(v + w)$ modular exponentiations with short exponents and w modular multiplications. In practice, one can select $|p| = 1024$ or $|p| = 2048$, and $|q| = 160$ or $|q| = 224$. Communication overhead amounts to $2(v + 1)$ $|p|$-bit and w κ-bit values.

Definitions and Assumptions

Semi-Honest Participants. We start with security in the semi-honest model. Security in the malicious model is treated in Section 3.2. Note that the term *adversary* refers to insiders, i.e., protocol participants. Outside adversaries are not considered, since their actions can be mitigated via standard network security techniques.

DDH Assumption. Let $\mathbb{G}$ be a cyclic group and g be its generator. We assume that bit-length of group size is l. The DDH problem is hard in $\mathbb{G}$ if, for every efficient algorithm $\mathcal{A}$, the following probability is a negligible function of κ:

$$\left|\Pr[x, y \leftarrow \{0,1\}^l : \mathcal{A}(g, g^x, g^y, g^{xy}) = 1] - \Pr[x, y, z \leftarrow \{0,1\}^l : \mathcal{A}(g, g^x, g^y, g^z) = 1]\right|$$

One-More-DH Assumption. Informally, the One-More-DH assumption [4] states that the DH problem is hard even if the adversary is given access to a "DH" oracle. Formally, let $(\mathbb{G}, q, g) \leftarrow KeyGen(\kappa)$ the Key-Generation algorithm outputting a multiplicative group of order q and assume $x \leftarrow \mathbb{Z}_q$. We say that the One-More-DH problem is (τ, t)-hard if for every algorithm $\mathcal{A}$ that runs in time t we have:

$$Pr[\{(g_i, (g_i)^x)\}_{i=1,\cdots v+1} \leftarrow \mathcal{A}^{DH_x(\cdot)}(g_1, \cdots, g_{ch})] \leq \tau$$

where $ch > v$ and $\mathcal{A}$ made at most v queries to the $DH_x(\cdot)$ oracle.

Gap-One-More-DH Assumption [22]. We say that that Gap-One-More-DH problem is hard if One-More-DH is hard even when the adversary has access to a DDH oracle. Formally,

$$Pr[\{(g_i, (g_i)^x)\}_{i=1,\cdots v+1} \leftarrow \mathcal{A}^{DH_x(\cdot), DDH(\cdot,\cdot,\cdot,\cdot)}(g_1, \cdots, g_{ch})] \leq \tau$$

where $ch > v$ and $\mathcal{A}$ made at most v queries to the $DH_x(\cdot)$ oracle.

For simplicity, we replace the oracle $DDH(\cdot,\cdot,\cdot,\cdot)$ with $DL_x(\cdot,\cdot)$ that, on input (a, b), returns 1 iff $b = a^x$. The oracle $DL_x(\cdot,\cdot)$ can be easily constructed from $DDH(\cdot,\cdot,\cdot,\cdot)$ and $DH_x(\cdot,\cdot)$ as $DL_x(\cdot,\cdot) = DDH(a, DH_x(a), \cdot, \cdot)$.

Definition 3 (Correctness). *If both parties are honest, at the end of the protocol, executed on on inputs $((\mathcal{S}, v), (\mathcal{C}, w))$, server outputs $\perp$, and client outputs $(|\mathcal{S} \cap \mathcal{C}|)$.*

The following client and server privacy definitions follow from those in related work [11, 10, 14]. In particular, as formalized in [13] (Sec. 7.2.2), in case of semi-honest parties, the traditional "real-versus-ideal" definition framework is *equivalent* to a much simpler framework that extends the formulation of honest-verifier zero-knowledge. Informally, a protocol privately computes certain functionality if whatever can be obtained from one party's view of a protocol execution can be obtained from input and output of that party. In other words, the view of a semi-honest party (including $\mathcal{C}$ or $\mathcal{S}$, all messages received during execution, and the outcome of that party's internal coin tosses), on each possible input $(\mathcal{C}, \mathcal{S})$, can be efficiently simulated considering only that party's own input and output.

Definition 4 (Client Privacy). *Let $\mathsf{View}_S(\mathcal{C}, \mathcal{S})$ be a random variable representing server's view during execution of PSI-CA with inputs $\mathcal{C}, \mathcal{S}$. There exists a* PPT *algorithm S^* such that:*

$$\{S^*(\mathcal{S}, |\mathcal{S} \cap \mathcal{C}|)\}_{(\mathcal{C},\mathcal{S})} \stackrel{c}{\equiv} \{\mathsf{View}_S(\mathcal{C}, \mathcal{S})\}_{(\mathcal{C},\mathcal{S})}$$

Definition 5 (Server Privacy). *Let $\mathsf{View}_C(\mathcal{C}, \mathcal{S})$ be a random variable representing client's view during execution of PSI-CA with inputs $\mathcal{C}, \mathcal{S}$. There exists a* PPT *algorithm C^* such that:*

$$\{C^*(\mathcal{C}, |\mathcal{S} \cap \mathcal{C}|)\}_{(\mathcal{C},\mathcal{S})} \stackrel{c}{\equiv} \{\mathsf{View}_C(\mathcal{C}, \mathcal{S})\}_{(\mathcal{C},\mathcal{S})}$$

In other words, on each possible pair of inputs $(\mathcal{C}, \mathcal{S})$, client's view can be efficiently simulated by C^* on input: $\mathcal{C}$ and $|\mathcal{S} \cap \mathcal{C}|$ (as well as v, w). Thus, as in [13], we claim that the two distributions implicitly defined above are computationally indistinguishable.

Proofs

Correctness. For any c_i held by client and s_j held by server, if $c_i = s_j$, hence, $hc_i = hs_j$, we obtain:

$$tc_{\ell_i} = H'(bc_i) = H'(Y^{R_c} \cdot {a_{\ell_i}}^{(1/R'_c)}) = H'(g^{R_c R_s} \cdot {hs_j}^{R'_s})$$
$$ts_j \quad = H'(bs_j) = H'(X^{R_s} \cdot hs_j^{R'_s}) \quad = H'(g^{R_c R_s} \cdot {hs_j}^{R'_s})$$

Hence, client learns set intersection cardinality by counting the number of matching pairs (ts_j, tc_{ℓ_i}). □

Client Privacy. We claim that the views of server – i.e., $\mathcal{S}$, X and $a_i = H(c_i)^{R'_c}$ for $i = 1, \ldots, v$ where H is modeled as a random oracle – is indistinguishable from $r_0, \ldots r_v$ with $r_i \leftarrow \mathbb{Z}_p$. Therefore it is trivial to construct a PPT algorithm S^* such that $\{S^*(\mathcal{S}, |\mathcal{S} \cap \mathcal{C}|)\}_{(\mathcal{C},\mathcal{S})} \stackrel{c}{\equiv} \{\mathsf{View}_S(\mathcal{C}, \mathcal{S})\}_{(\mathcal{C},\mathcal{S})}$.

X is distributed identically to r_0 since R_c is uniformly distributed in $\mathbb{Z}_q$. As such, we omit it from the rest of the proof.

When $v = 1$, for any $hc_1 = H(c_1)$ there exists R_{c_1} such that $a_1 = hc_1^{R_{c_1}}$. Therefore, a_1 is uniformly distributed.

For $v \geq 2$ $a_1, \ldots a_v$ is indistinguishable from $r_1, \ldots r_v$ assuming the hardness of DDH. In particular, the existence of an efficient distinguisher $\mathcal{D}$ that outputs 0 when presented with $r_1, \ldots r_v$ and outputs 1 when it observes $a_1, \ldots a_v$ allows us to construct a simulator $\overline{\mathsf{SIM}}_s$ that violates the DDH assumption, as follows.

Upon receiving a DDH challenge $(\bar{g}, \bar{g}^x, \bar{g}^y, \bar{g}^z)$, $\overline{\mathsf{SIM}}_s$:

- Selects a random set $\overline{\mathcal{C}}$ composed of v elements $\overline{\mathcal{C}} = \{c_1, \ldots, c_v\}$, $v-2$ random values $r_1, \ldots, r_{v-2}$ from $\mathbb{Z}_q$ and R_c at random from $\mathbb{Z}_q$.
- Sends $\{\overline{a}_1, \ldots, \overline{a}_v\} = \{\bar{g}^y, \bar{g}^z, (\bar{g}^y)^{r_1}, \ldots, (\bar{g}^y)^{r_{v-2}}\}$ to $\mathcal{D}$.
- Answers queries for H as follows: $H(c_1) = \bar{g}$; $H(c_2) = \bar{g}^x$; $H(c_i) = \bar{g}^{r_{i-2}}$ for $3 \leq i \leq v$ and with a random value otherwise. Queries and responses to H are stored by $\overline{\mathsf{SIM}}_S$ for consistency.

Note that if $(\bar{g}, \bar{g}^x, \bar{g}^y, \bar{g}^z)$ is a Diffie-Hellman tuple, i.e. $z = xy$, then $\overline{a}_1, \ldots, \overline{a}_v$ is distributed like $a_1, \ldots, a_v$; thus, $\mathcal{A}_1$ must output 1. If $(\bar{g}, \bar{g}^x, \bar{g}^y, \bar{g}^z)$ is not a Diffie-Hellman tuple, then $\overline{a}_1, \ldots, \overline{a}_v$ is not properly distributed (since $a_2 \neq (H(c_2))^y$) and therefore $\mathcal{D}$ must output 0. As a result, $\overline{\mathsf{SIM}}_s$ can use $\mathcal{D}$'s output to respond to the DDH challenge correctly iff $\mathcal{A}_1$'s output is correct. Therefore, $\mathcal{D}$ can only answer correctly with negligible advantage over random guessing.

By applying the hybrid argument, it is easy to see that gradually replacing one value a_i at a time with a random value in $\mathbb{Z}_p$, $\mathcal{D}$ has only negligible advantage in detecting such change. □

Server Privacy. We show that client's view can be efficiently simulated by a PPT algorithm SIM_C, i.e., $\{\mathsf{SIM}_C(\mathcal{C}, |\mathcal{S} \cap \mathcal{C}|)\}_{(\mathcal{C},\mathcal{S})} \stackrel{c}{\equiv} \{\mathsf{View}_C(\mathcal{C}, \mathcal{S})\}_{(\mathcal{C},\mathcal{S})}$. The simulator is constructed as follows:

1. SIM_C builds two tables $T_1 = (u, h)$ and $T_2 = (u', h')$ to answer the H and H' queries respectively, and initializes an empty set V. SIM_C responds to a query u (resp. u') with a value in $h \leftarrow \mathbb{Z}_p$ for H ($h' \leftarrow \mathbb{Z}_p$ for H'), and stores (u, h) in T_1 ((u', h') in T_2 resp.). SIM_C uses T_1, T_2 to respond consistently to queries from client.
2. SIM_C constructs a set $TS = \{ts_1, \ldots, ts_w\}$, where $ts_i \leftarrow \mathbb{Z}_p$, and a random subset $TS' = \{ts'_1, \ldots, ts'_{|\mathcal{I}|}\} \subseteq TS$, such that $|TS'| = |\mathcal{I}|$.
3. Upon receiving $X, \{a_1, \ldots, a_v\}$ from client, SIM_C picks $R_s \leftarrow \mathbb{Z}_q$ and $R'_s \leftarrow \mathbb{Z}_q$, computes $a'_i = a_i^{R'_s}$ and adds $(a', (a')^{R'_s})$ to V. Finally SIM_C sends $Y = X^{Rs}$, $\Pi(a'_i, \ldots, a'_v)$ and $\{ts_1, \ldots, ts_w\}$ to client.
4. SIM_C adds $|\mathcal{I}|$ pairs $((X^{R_s} \cdot H(c_i)^{R'_s}, ts'_i \in TS')$ to T_2 and continues to answer queries to H and H' consistently using T_1 and T_2 as defined in Step 1.
5. SIM_C answers queries to H and H' as follows:
   (a) For query u to H never posed before, SIM_C responds with a value in $h \leftarrow \mathbb{Z}_p$ and adds (u, h) to T_1.
   (b) For query u' to H' never asked before, if there is a pair (u, h) in T_1, such that $h^{R'_s} = (u'/X^{R_s})$, add $(h, (u'/X^{R_s}))$ to table V and aborts; otherwise SIM_C responds with $h' \leftarrow \mathbb{Z}_p$ and adds (u', t') to T_2.

Any efficient semi-honest client C^* cannot distinguish between an interaction with an honest server and SIM_C. By construction, C^*'s view differs from the interaction with an honest server only when SIM_C aborts. In particular, the existence of an efficient C^* that can cause SIM_C to abort allows us to construct an efficient adversary for the Gap-One-More-DH problem. The reduction $\overline{\mathsf{SIM}}_C$ is constructed modifying SIM_C as follows:

- Given a Gap-One-More-DH challenge $\mathsf{Ch} = (g_1, \ldots, g_{ch})$ and access to oracles $(\cdot)^x$ and $DL_x(\cdot, \cdot)$, $\overline{\mathsf{SIM}}_C$ responds to a query u for H, where u was never asked before, with a fresh g_i rather than with a random element in $\mathbb{Z}_p$ as defined in Step 1 above.
- In Step 3, upon receiving $X, \{a_1, \ldots, a_v\}$ from C^*, $\overline{\mathsf{SIM}}_C$ picks $R_s \leftarrow \mathbb{Z}_q$ and uses the oracle $(\cdot)^x$ to compute $a'_i = a_i^x$ and adds (a', a'^x) to V (i.e. $\overline{\mathsf{SIM}}_C$ invokes the oracle $(\cdot)^x$ exactly v times). Then SIM_C sends $Y = X^{Rs}$, $\{ts_1, \ldots, ts_w\}$ and $\Pi(a'_i, \ldots, a'_v)$ to C^*.
- In Step 4, $\overline{\mathsf{SIM}}_C$ does not add any new pair to T_2 and simply answers queries to H and H' consistently.
- In Step 5.(a), $\overline{\mathsf{SIM}}_C$ responds to a new query u for H with a fresh element $g_i \in \mathsf{Ch}$.
- In Step 5.(b), upon receiving a query u' for H', $\overline{\mathsf{SIM}}_C$ computes $t = u'/X^{R_s}$. If there exists an element $g_i \in \mathsf{Ch}$ such that $DL_x(g_i, t) = 1$ and $(g_i, t) \notin V$, then $\overline{\mathsf{SIM}}_C$ adds (g_i, t) to V.

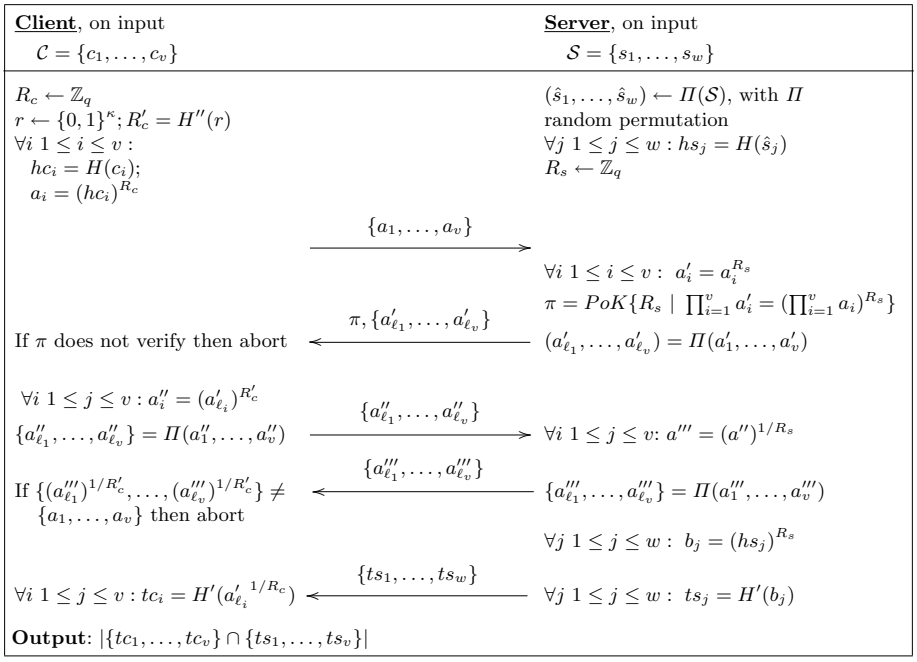

Figure 2: Proposed PSI-CA Protocol secure in the malicious model. All computation is mod p.

At the end of the execution, we have that $|V| > v$. $\overline{\mathsf{SIM}}_C$ can now use $|V|$ to answer the Gap-One-More-DH challenge. Since this contradicts the hardness of the Gap-One-More-DH problem, SIM_C only aborts – and therefore C^* detects the simulation – with negligible probability. □

3.2 Security Against Malicious Server

In Figure 2, we present a PSI-CA variant secure against malicious server and semi-honest client. It is executed on common input of two primes p, q, s.t., $q|p-1$, a generator g of a subgroup of size q, and three hash functions (modeled as random oracles), $H : \{0,1\}^* \to \mathbb{Z}_p^*$, $H' : \{0,1\}^* \to \{0,1\}^\kappa$ and $H'' : \{0,1\}^* \to \mathbb{Z}_q^*$, given the security parameter κ.

Compared to the semi-honest secure construction in Figure 1, this protocol introduces two additional communication rounds. Nonetheless, protocol complexities are only slightly affected – in fact, they remain linear in $(w + v)$. The extra rounds are needed to prevent a malicious server from using a different exponent when computing the $a'_{\ell_i} = a^{R_s}$ values. Observe that replacing the proof of knowledge (computed by server) with a proof of logical *and* of v separate statements "$a'_i = a_i^{R_s}$" would not yield a malicious-secure PSI-CA construct. Intuitively, such proof would unfortunately reveal the relationship between each index i and corresponding index ℓ_i, thus, allow client to determine *which* elements belong to the intersection, rather than just "*how many*".

Definition 6 (Client Privacy in Malicious Model). *For every* PPT S^* *that plays the role of server, there exists an algorithm* $\overline{S^*}$ *that is admissible in the ideal model (as of Definition 7.2.4 of [13]) such that:*

$$\{\mathsf{IDEAL}_{\text{PSI-CA},(\overline{S^*},C)}(\mathcal{S},\mathcal{C})\}_{\mathcal{C},\mathcal{S}} \stackrel{c}{\equiv} \{\mathsf{REAL}_{\text{PSI-CA},(S^*,C)}(\mathcal{S},\mathcal{C})\}_{\mathcal{C},\mathcal{S}}$$

Theorem 1. *If the Gap-One-More-DH problem is hard and the proof system* π *is a zero-knowledge proof of knowledge, the protocol in Figure 2 is a secure computation of PSI-CA in ROM, as per definitions 2, 3, 5, and 6.*

Proof. We start with proving correctness, then, we build an ideal-world simulator from a malicious real-world server, and, finally, we show that client's view can be efficiently simulated by a PPT algorithm.

Correctness. For any c_i held by client and s_j held by server, if $c_i = s_j$, hence, $hc_i = hs_j$, we obtain:

$$\begin{aligned} tc_i &= H'((a'_i)^{1/R_c}) = H'(hs_j{}^{R_s}) \\ ts_j &= H'(b_j) \quad = H'(hs_j{}^{R_s}) \end{aligned}$$

Hence, client learns set intersection cardinality by counting the number of matching pairs (ts_j, tc_i). □

Building an ideal-world simulator SIM_S *from malicious real-world Server* S^*. We now show how to construct a simulator SIM_S that uses a malicious real-world server to interact with a trusted third party in the ideal world. We also construct a simulator SIM_C that uses a semi-honest client.SIM_S is constructed as follows:

1. SIM_S replies to H, H' and H'' queries with random values, and keeps track of its replies in tables T, T' and T'' respectively for consistency.
2. SIM_S picks a set of v random elements $a_i \leftarrow \mathbb{Z}_p$ and sends them to S^*.
3. S^* sends π, a'_i to SIM_S.
4. SIM_S interacts with S^* as verifier in proof π. If π does not verify, SIM_S aborts; otherwise it runs the extractor algorithm for π and obtains R_s.
5. Then SIM_S picks a random $r \leftarrow \{0,1\}^\kappa$, computes $R'_c = H''(r)$ and returns $a'' = (a'_i)^{R'_c}$ to S^*.
6. SIM_S checks whether $\{(a'''_1)^{1/R'_c}, \ldots, (a'''_v)^{1/R'_c}\} \stackrel{?}{=} \{a_1, \ldots, a_v\}$. If not, it aborts.
7. For each ts_i received from S^*, SIM_S checks if there exist a pair $(u', h') \in T'$ such that $h' = ts_i$. If such pair exists, it computes $hs_i = (u')^{1/R_s}$; otherwise, it adds a dummy to set $\mathcal{S}'$. Then SIM_S checks if there is a pair (u, h) with $h = hs_i$, and adds the corresponding value u to $\mathcal{S}'$; if there is no such pair, SIM_S adds a dummy to set $\mathcal{S}'$.
8. Finally, SIM_S uses set $\mathcal{S}'$ as its input with the trusted third party in the ideal world.

We claim that the views of S^* (i.e., $\mathcal{S}$ and $a_i = H(c_i)^{R_c}$ for $i = 1, \ldots, v$ where H is modeled as a random oracle) when interacting with SIM_S and with real C are computationally indistinguishable. Moreover, SIM interacts with the TTP in the ideal world with S^*'s input.

When $v = 1$, for any $hc_1 = H(c_1)$ there exists R_{c_1} s.t. $a_1 = hc_1^{R_{c_1}}$. Thus, a_1 is distributed exactly as expected by S, which cannot detect the simulation.

For $v \geq 2$ the simulation is undetectable assuming the hardness of DDH. In particular, the existence of an efficient algorithm $\mathcal{A}_1$ that detects the simulation (i.e., outputs 0 when interacting with SIM_S and 1 when it interacts with a honest client) allows us to construct a reduction to the DDH problem, as follows.

Upon receiving a DDH challenge $(\bar{g}, \bar{g}^x, \bar{g}^y, \bar{g}^z)$, $\overline{\mathsf{SIM}}_s$:

- Selects a random set $\overline{\mathcal{C}}$ composed of v elements $\overline{\mathcal{C}} = \{c_1, \ldots, c_v\}$, $v-2$ random values $r_1, \ldots, r_{v-2}$ from $\mathbb{Z}_q$ and R_c at random from $\mathbb{Z}_q$.
- Sends $\{a_1, \ldots, a_v\} = \{\bar{g}^y, \bar{g}^z, (\bar{g}^y)^{r_1}, \ldots, (\bar{g}^y)^{r_{v-2}}\}$ to $\mathcal{A}_1$.
- Answers queries for H as follows: $H(c_1) = \bar{g}$; $H(c_2) = \bar{g}^x$; $H(c_i) = \bar{g}^{r_{i-2}}$ for $3 \leq i \leq v$ and with a random value otherwise. Queries and responses to H are stored by $\overline{\mathsf{SIM}}_S$ for consistency.

Note that if $(\bar{g}, \bar{g}^x, \bar{g}^y, \bar{g}^z)$ is a Diffie-Hellman tuple, i.e. $z = xy$, then $\{a_1, \ldots, a_v\}$ is distributed exactly like the output of a honest client; therefore $\mathcal{A}_1$ must output 1. If $(\bar{g}, \bar{g}^x, \bar{g}^y, \bar{g}^z)$ is not a Diffie-Hellman tuple, then $\{a_1, \ldots, a_v\}$ is not properly distributed (since $a_2 \neq (H(c_2))^y$) and therefore $\mathcal{A}_1$ must output 0. As a result, $\overline{\mathsf{SIM}}_s$ can use $\mathcal{A}_1$'s output to respond to the DDH challenge correctly iff $\mathcal{A}_1$'s output is correct. Therefore, $\mathcal{A}_1$ can only answer correctly with negligible advantage over random guessing.

By applying the hybrid argument, it is easy to see that gradually replacing one value a_i at a time with a random value in $\mathbb{Z}_p$, $\mathcal{A}_1$ has only negligible advantage in detecting such change. By replacing all values a_i with random elements in $\mathbb{Z}_p$ we obtain the output distribution of SIM_C. Therefore S^* can detect the simulation with only negligible probability.

We also claim that (1) S^* must use the same R_s to compute all values a'_i, i.e., $a'_i = a_i^{R_s}$ for $1 \leq i \leq v$ to later return $(a'_i)^{R'_c/R_s}$, and (2) π proves the knowledge of the *same* R_s used to compute values a'_i. This is required to prove that the set $\mathcal{S}'$ is correctly constructed by SIM_S, i.e., it corresponds to S^*'s input.

To prove (1), let us define algorithm $\mathcal{A}_2$ that interacts in a four-round protocol (which corresponds to the first four rounds of the protocol in Figure 2, except that $\mathcal{A}_2$ is allowed to use different exponents to compute values a'_i) with a challenger $C_{\mathcal{A}_2}$ as follows:

1. $\mathcal{A}_2$ receives $a_1, \ldots, a_v \in \mathbb{Z}_p$ from $C_{\mathcal{A}_2}$.
2. $\mathcal{A}_2$ then computes $a'_i = a_i^{R_{si}}$ and returns $\pi, a'_{\ell_1}, \ldots, a'_{\ell_v} = \Pi(a'_1, \ldots, a'_v)$ to $C_{\mathcal{A}_2}$ where $\pi = PoK\{R_s | \prod_i a_i^{R_{si}} = \prod_i a_i^{R_s}\}$.
3. $C_{\mathcal{A}_2}$ acts as verifier for π. If the verification fails, $C_{\mathcal{A}_2}$ aborts; otherwise it computes $\Pi((a'_1)^{R'_c}, \ldots, (a'_v)^{R'_c})$ and sends it to $\mathcal{A}_2$.
4. After receiving $\Pi((a'_1)^{R'_c}, \ldots, (a'_v)^{R'_c})$, $\mathcal{A}_2$ returns $\Pi((a_1)^{R'_c}, \ldots, (a_v)^{R'_c})$.

We claim that $\mathcal{A}_2$ is an efficient algorithm iff $R_{si} = R_{sj}$ for all i, j. Assume that there exists an efficient algorithm $\mathcal{A}_2$ as above, such that $R_{si} \neq R_{sj}$ for at least one pair of values i, j.

In particular, we observe that $\mathcal{A}_2$ cannot know both R_s as proven in π and the value of all exponents R_{si} such that $\prod_i a_i^{R_{si}} = \prod_i a_i^{R_s}$: assume the existence

of an efficient algorithm $\mathcal{A}_3$ that, given a_1, a_2, it returns R_{s1}, R_{s2}, R_{s3} such that $a_1^{R_{s1}} \cdot a_2^{R_{s2}} = (a_1 \cdot a_2)^{R_{s3}}$.[1] $\mathcal{A}_3$ can be used to efficiently compute discrete logarithm in our settings: if we write $a_1 = g$ and $a_2 = g^x$ for some (unknown) x, we have that $x = (R_{s3} - R_{s1})/(R_{s2} - R_{s3})$.

We point out that the knowledge of all values R_{si} are required to answer correctly (although with relatively small probability due to C's shuffle) to queries from client. Therefore S^* must use the same exponent for all exponentiations in order to be able to answer to the subsequent query from SIM_S.

Since S^* uses only one exponent, computing π requires the knowledge of the same value R_s used for values a'_i, i.e., this proves (2). For this reason, the extraction of S^* input is performed correctly by SIM_S. □

We now look at the output of the honest client in the real world interacting with SIM_S. SIM_S sends $\mathcal{S}'$ to trusted third party, which outputs the number of matching elements between $\mathcal{S}'$ and the set from the ideal world client. Since H, H' are modeled as random oracles, this output is identical to the output of the real world client that interacts with S^*.

Simulating client's view. We show that client's view can be efficiently simulated by a PPT algorithm SIM_C, i.e., $\{\mathsf{SIM}_C(\mathcal{C}, |\mathcal{S} \cap \mathcal{C}|)\}_{(\mathcal{C},\mathcal{S})} \stackrel{c}{\equiv} \{\mathsf{View}_C(\mathcal{C}, \mathcal{S})\}_{(\mathcal{C},\mathcal{S})}$. The simulator is constructed as follows:

1. SIM_C builds two tables $T = (u, h)$ and $T' = (u', h')$ to answer the H and H' queries respectively, and initializes an empty set V. SIM_C responds to a query u (resp. u') with a value in $h \leftarrow \mathbb{Z}_p$ for H ($h' \leftarrow \mathbb{Z}_p$ for H'), and stores (u, h) in T ((u', h') in T' resp.). SIM_C uses T, T' to respond consistently to the queries from client.
2. SIM_C constructs a set $TS = \{ts_1, \ldots, ts_w\}$, where $ts_i \leftarrow \mathbb{Z}_p$, and a random subset $TS' = \{ts'_1, \ldots, ts'_{|\mathcal{I}|}\} \subseteq TS$ such that $|TS'| = |\mathcal{I}|$.
3. Upon receiving $\{a_1, \ldots, a_v\}$ from client, SIM_C picks $R_s \leftarrow \mathbb{Z}_q$, computes $a'_i = a_i^{R_s}$ and adds $(a', (a')^{R_s})$ to V. It also compute π over R_s.
4. Upon receiving a''_i, SIM_C returns $a'''_i = (a''_i)^{1/R_s}$
5. SIM_C adds $|\mathcal{I}|$ pairs $((X^{R_s} \cdot H(c_i)^{R'_s}, ts'_i \in TS')$ to T'.
6. SIM_C answers queries to H and H' as follows:
   (a) For query u to H never asked before, SIM_C responds with a value in $h \leftarrow \mathbb{Z}_p$ and adds (u, h) to T.
   (b) For query u' to H' never asked before, if there is a pair (u, h) in T such that $h^{R'_s} = (u'/X^{R_s})$, add $(h, (u'/X^{R_s}))$ to table V and aborts; otherwise SIM_C responds with $h' \leftarrow \mathbb{Z}_p$ and adds (u', t') to T'.

Any efficient semi-honest client C^* cannot distinguish between the interaction with an honest server and SIM_C. By construction, C^*'s view differs from the interaction with an honest server only when SIM_C aborts. In particular, the

[1] Although, for the sake of simplicity, we define $\mathcal{A}_3$ with two elements in input, it is easy to see how the same argument applies for an adversary $\mathcal{A}'_3$ with input of size greater than two.

existence of an efficient C^* that can cause SIM_C to abort allows us to construct an efficient adversary for the Gap-One-More-DH problem. The reduction $\overline{\mathsf{SIM}}_C$ is constructed modifying SIM_C as follows:

- Given a Gap-One-More-DH challenge $\mathsf{Ch} = (g_1, \ldots, g_{ch})$ and access to oracles $(\cdot)^x$ and $DL_x(\cdot,\cdot)$, $\overline{\mathsf{SIM}}_C$ responds to a query u for H, where u was never asked before, with a fresh g_i rather than with a random element in $\mathbb{Z}_p$ as defined in Step 1 above.
- In Step 3, upon receiving $\{a_1, \ldots, a_v\}$ from C^*, $\overline{\mathsf{SIM}}_C$ picks $R_s \leftarrow \mathbb{Z}_q$ and uses the oracle $(\cdot)^x$ to compute $a'_i = a_i^x$ and adds (a', a'^x) to V (i.e. $\overline{\mathsf{SIM}}_C$ invokes the oracle $(\cdot)^x$ exactly v times). Then SIM_C sends $\{ts_1, \ldots, ts_w\}$, $\Pi(a'_i, \ldots, a'_v)$ and π to C^*, and simulates the proof for π. Since the proof system π is zero-knowledge, C^* cannot distinguish the real prover and the simulation except with negligible probability.
- In Step 4, $\overline{\mathsf{SIM}}_C$ determines for which tuple $(u'', h'') \in T''$ it holds
$$\{(a'_1)^{h''}, \ldots, (a'_v)^{h''}\} = \{a''_1, \ldots, a''_v\}$$
and sets $R'_c = h''$; then it returns $\Pi((a_1)^{R'_c}, \ldots, (a_v)^{R'_c})$ to C^*.
- In Step 5, $\overline{\mathsf{SIM}}_C$ does not add any new pair to T' and simply answers queries to H and H' consistently.
- In Step 6.(a), $\overline{\mathsf{SIM}}_C$ responds to a new query u for H with a fresh element $g_i \in \mathsf{Ch}$.
- In Step 6.(b), upon receiving a query u' for H', $\overline{\mathsf{SIM}}_C$ computes $t = u'/X^{R_s}$. If there exists an element $g_i \in \mathsf{Ch}$ such that $DL_x(g_i, t) = 1$ and $(g_i, t) \notin V$, then $\overline{\mathsf{SIM}}_C$ adds (g_i, t) to V.

At the end of the execution, we have that $|V| > v$. $\overline{\mathsf{SIM}}_C$ can now use $|V|$ to answer the Gap-One-More-DH challenge. Since this contradicts the hardness of the Gap-One-More-DH problem, SIM_C only aborts – and therefore C^* detects the simulation – with negligible probability. □

4 Fast Authorized PSI-CA

We now introduce the concept of Authorized PSI-CA (APSI-CA). It extends PSI-CA to enforce *authorization* of client input. Similar to APSI [9] (reviewed in Section 2), APSI-CA involves an offline trusted third party – Certification Authority (CA) – that provides client with authorizations (in practice, signatures) to input into the set intersection cardinality computation.

Definition 7 (Authorized PSI-CA (APSI-CA)). *A protocol involving a server, on input of a set of w items: $\mathcal{S} = \{s_1, \cdots, s_w\}$, and a client, on input of a set of v items with associated authorizations (i.e., signatures), $\mathcal{C} = \{(c_1, \sigma_i) \cdots, (c_v, \sigma_v)\}$. It results in client outputting $|\boldsymbol{\mathcal{I}^*}|$, where:*

$$\mathcal{I}^* = \{s_j \in \mathcal{S} \mid \exists (c_i, \sigma_i) \in \mathcal{C} \ s.t. \ c_i = s_j \wedge \mathsf{Verify}(\sigma_i, c_i) = 1\}.$$

APSI-CA entails the following informal privacy requirements:

- *Server Privacy (APSI-CA).* The client learns no information beyond what can be inferred from the protocol output, i.e., (1) cardinality of set intersection on authorized items and (2) upper bound on the size of $\mathcal{S}$.

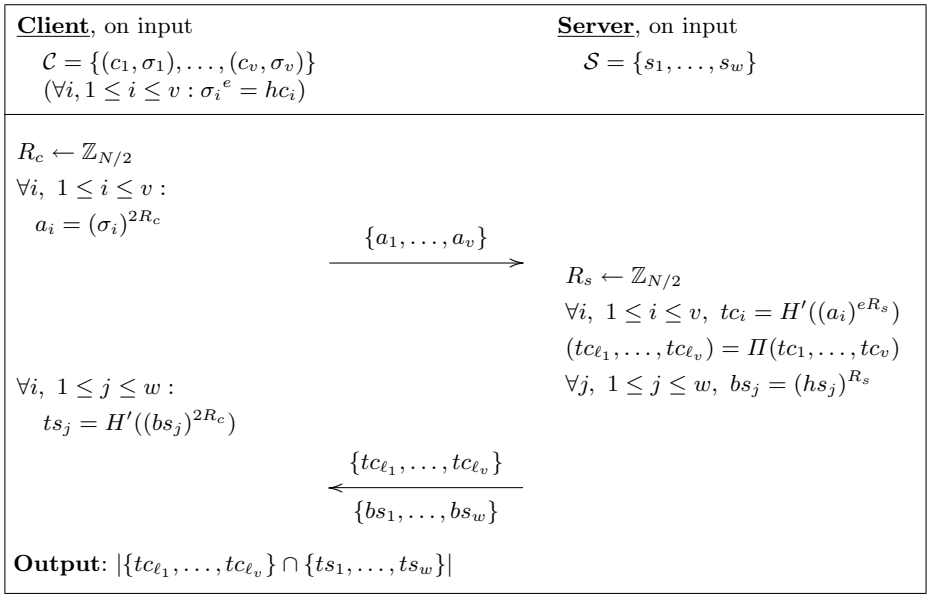

Figure 3: Authorized PSI-CA. All computation is mod N.

- *Client Privacy (APSI-CA).* No information is leaked about items or authorizations in client set (except an upper bound on their number).
- *Unlinkability.* Similar to PSI-CA, we require that neither server nor client can determine if any two instances of the protocol are related, i.e., executed on the same input by client or server.

We illustrate our APSI-CA protocol in Figure 3. Note that the CA is responsible for generating all public parameters: on input the security parameter κ, it executes $(N, e, d, g) \leftarrow RSA.KGen(\kappa)$, where g is a generator of QR_N, and selects $H : \{0,1\}^* \rightarrow {\mathbb{Z}_N}^*$ (Full-Domain Hash) and $H' : \{0,1\}^* \rightarrow \{0,1\}^\kappa$ (random oracles). The CA authorizes client input c_i by issuing $\sigma_i = H(c_i)^d \bmod N$ (i.e., an RSA signature). The protocol is executed between client and server, on common input (N, e, H, H'). We assume that server's input $(\mathcal{S})$ is randomly permuted before protocol execution to mask any *ordering* of the items contained in it. Finally, observe that hc_i and hs_j denote, respectively, $H(c_i)$ and $H(s_j)$.

Similar to its PSI-CA counterpart, this APSI-CA has the following properties:

- ***Correctness.*** For any (σ_i, c_i) held by client and s_j held by server, if: (1) σ_i is a genuine CA signature on c_i, and (2) $c_i = s_j$, hence, $hc_i = hs_j$, we obtain: $tc_{\ell_i} = H'((\sigma_i)^{2eR_cR_s}) = H'((hc_i)^{2R_cR_s}) = ts_j$.
- ***Privacy.*** In this version of the paper, we only provide some intuition for our security arguments, and defer to future work formal proofs. Client privacy is based on its input being statistically indistinguishable from a random distribution in QR_N. Arguments regarding server privacy are similar to those for PSI-CA, thus, we do not repeat them here. We argue that if one could violate APSI-CA server privacy, then the one would also violate server pri-

vacy of the APSI construct in Fig.1 of [8], proven secure under the RSA and DDH assumptions. Finally, note that the protocol is unlinkable, given that random values, R_c, R_s, are selected fresh for each protocol execution.

- ***Efficiency.*** This APSI-CA protocol incurs linear computation (for both parties) and communication complexity. Specifically, client performs $O(w)$ modular exponentiations and server – $O(w + v)$. However, exponents are now taken in the RSA settings, while in PSI-CA can be taken from a smaller group, thus, be much shorter (e.g., 160-bit vs 1024-bit long). Communication complexity amounts to $O(w + v)$. Note that this is significantly lower than state of the art, i.e., [6], which incurs quadratic ($O(wv)$) communication and computation overhead.

5 Combining PSI-CA and PSI

As mentioned in Section 1, it is often desirable to privately assess the magnitude of the set intersection before engaging in an actual (private) set intersection computation. We are motivated by potential concerns with respect to server privacy, arising in PSI executions where the intersection obtained by client is close to the entire server set (i.e., $|\mathcal{S} \cap \mathcal{C}| \approx |\mathcal{S}|$).

We now show how to combine our proposed PSI-CA construct with with PSI, in order to address such concerns. Specifically, before engaging in PSI, parties first run the PSI-CA protocol with their client/server roles reversed. This way, server learns the intersection cardinality and then uses it to *decide* whether to to proceed with PSI. In other words, server defines a *policy* – based on the size of (i) the two sets and (ii) the intersection – and only if the policy is satisfied, server engages in PSI protocol (thus, client privately obtains the set intersection).

The resulting protocol is presented in Figure 4. In the first two rounds, server and client run PSI-CA with their *roles reversed* (i.e., server learns the cardinality of the intersection), and, assuming server's policy is satisfied, the last round allows client to learn the set intersection. The same approach can be used for other private set operations, such as PSU [15]. Indeed, similar concerns about server privacy occur in a scenario where $|\mathcal{C} \cup \mathcal{S}| \approx |\mathcal{C}| + |\mathcal{S}|$, and can again be addressed by running PSI-CA with roles reversed.

The security of this protocol, in presence of semi-honest adversaries, trivially stems from that of the underlying PSI-CA. Nonetheless, we defer to future work extending our constructions to malicious security. In fact, there is no guarantee that malicious parties maintain the same input over multiple interactions, thus, realizing a malicious-secure version of our protocol in Figure 4 remains an interesting open problem.

Remark: Our technique in Figure 4 is not to be confused with the concept of Policy-Enhanced PSI, recently proposed by [33]. Using the latter, two parties privately obtain the intersection of their sets, while enforcing policies pertaining what/how to share, based on policies and authorizations related to single items. Whereas, policy enforced by server in our protocol is based on the cardinality of set intersection: depending on this (and on its relationship to set size), server decides whether or not to disclose set intersection's content to client. A vaguely

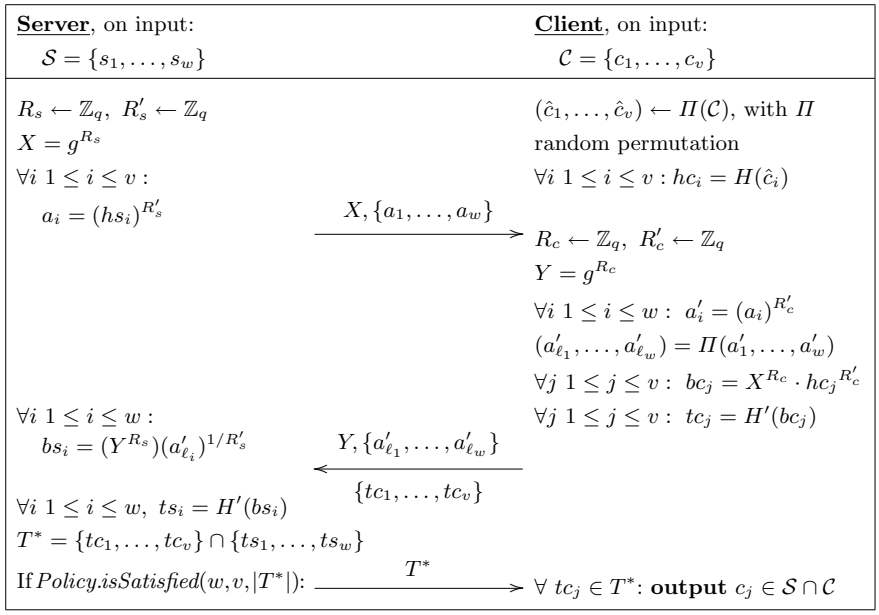

Figure 4: Combining PSI-CA and PSI for a three-round "policy-based" Private Set Intersection protocol. (All computation is mod p).

comparable approach is so-called Knowledge-oriented Multi-party Secure Computation [26], where each participating party is able to reason about the increase in knowledge that other parties could gain as a result of the secure computation, and may choose not to participate to restrict that gain.

6 Conclusion

This paper presented two protocols for PSI-CA: one secure with respect to semi-honest participants, and the other secure against malicious server. Both are appreciably more efficient than state of the art: they achieve *linear* computational and communication complexities, while prior work incurred *quadratic* complexities. We also showed how these protocols can be used to compute PSU-CA without introducing any additional overhead.

Further, we presented another protocol, namely, APSI-CA, which extends to settings where client input must be authorized by a certification authority. Finally, we used proposed PSI-CA construct to realize a PSI protocol where server determines (in privacy-preserving manner) cardinality of set intersection before deciding whether or not to engage in a PSI interaction with client.

As part of future work, we plan to look into malicious-secure and/or UC-secure *composition* of protocols. While our PSI-CA protocol is secure against malicious server, there is no guarantee that such party maintain the same input over multiple interactions, thus, for instance, realizing a malicious-secure version of our protocol in Figure 4 remains an interesting open challenge. Finally, we plan to release, alongside the final version of the paper, optimized implementations and extensive performance evaluation of proposed protocols.

References

1. B. Applebaum, H. Ringberg, M. Freedman, M. Caesar, and J. Rexford. Collaborative, Privacy-preserving Data Aggregation at Scale. In *PETS*, 2010.
2. G. Ateniese, M. Blanton, and J. Kirsch. Secret handshakes with dynamic and fuzzy matching. In *NDSS*, 2007.
3. P. Baldi, R. Baronio, E. De Cristofaro, P. Gasti, and G. Tsudik. Countering GATTACA: Efficient and Secure Testing of Fully-Sequenced Human Genomes. In *CCS*, 2011.
4. M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko. The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme. *Journal of Cryptology*, 16(3), 2003.
5. E. Bursztein, J. Lagarenne, M. Hamburg, and D. Boneh. OpenConflict: Preventing Real Time Map Hacks in Online Games. In *S&P*, 2011.
6. J. Camenisch and G. M. Zaverucha. Private intersection of certified sets. In *Financial Cryptography*, 2009.
7. E. De Cristofaro, S. Jarecki, J. Kim, and G. Tsudik. Privacy-preserving policy-based information transfer. In *PETS*, 2009.
8. E. De Cristofaro, J. Kim, and G. Tsudik. Linear-Complexity Private Set Intersection Protocols Secure in Malicious Model. In *Asiacrypt*, 2010.
9. E. De Cristofaro and G. Tsudik. Practical private set intersection protocols with linear complexity. In *Financial Cryptography*, 2010.
10. M. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In *TCC*, 2005.
11. M. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Eurocrypt*, 2004.
12. K. Frikken. Privacy-Preserving Set Union. In *ACNS*, 2007.
13. O. Goldreich. *Foundations of Cryptography*. Cambridge U. Press, 2004.
14. C. Hazay and Y. Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *TCC*, 2008.
15. C. Hazay and K. Nissim. Efficient Set Operations in the Presence of Malicious Adversaries. In *PKC*, 2010.
16. S. Hohenberger and S. Weis. Honest-verifier private disjointness testing without random oracles. In *PET*, 2006.
17. J. Hong, J. W. Kim, J. Kim, K. Park, and J. H. Cheon. Constant-Round Privacy Preserving Multiset Union. Cryptology ePrint Archive, Report 2011/138, 2011. `http://eprint.iacr.org/2011/138`.
18. Y. Huang, D. Evans, and J. Katz. Private Set Intersection: Are Garbled Circuits Better than Custom Protocols? In *NDSS*, 2012.
19. Intelligence Advanced Research Projects Activity (IARPA). Automatic Privacy Protection and Security And Privacy Assurance Research Programs. `https://www.fbo.gov/utils/view?id=920029a5107a9974c2e379324a1dcc4e`.
20. Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. *CRYPTO*, 2003.
21. S. Jarecki and X. Liu. Efficient Oblivious Pseudorandom Function with Applications to Adaptive OT and Secure Computation of Set Intersection. In *TCC*, 2009.
22. S. Jarecki and X. Liu. Fast secure computation of set intersection. In *SCN*, 2010.
23. M. Kantarcioglu, R. Nix, and J. Vaidya. An efficient approximate protocol for privacy-preserving association rule mining. *KDD*, 2009.
24. L. Kissner and D. Song. Privacy-preserving set operations. In *Crypto*, 2005.

25. M. Li, N. Cao, S. Yu, and W. Lou. Findu: Privacy-preserving personal profile matching in mobile social networks. In *INFOCOM*, 2011.
26. P. Mardziel, M. Hicks, J. Katz, and M. Srivatsa. Knowledge-oriented secure multiparty computation. In *PLAS*, 2012.
27. A. Menezes, P. V. Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC, 1997.
28. S. Nagaraja, P. Mittal, C. Hong, M. Caesar, and N. Borisov. BotGrep: Finding Bots with Structured Graph Analysis. In *Usenix Security*, 2010.
29. A. Narayanan, N. Thiagarajan, M. Lakhani, M. Hamburg, and D. Boneh. Location Privacy via Private Proximity Testing. In *NDSS*, 2011.
30. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Eurocrypt*, 1999.
31. S. Pohlig and M. Hellman. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. *IEEE Transactions on information Theory*, 24(1), 1978.
32. J. H. Seo, J. H. Cheon, and J. Katz. Constant-Round Multi-Party Private Set Union using Reversed Laurent Series. In *PKC*, 2012.
33. E. Stefanov, E. Shi, and D. Song. Policy-enhanced private set intersection: Sharing information while enforcing privacy policies. In *PKC*, 2012.
34. J. Vaidya and C. Clifton. Secure set intersection cardinality with application to association rule mining. *Journal of Computer Security*, 13(4), 2005.
35. A. Yao. Protocols for secure computations. In *FOCS*, 1982.